



# Programación Python para Big Data

## Lección 3: PyCaret y AutoML

# ÍNDICE

<b>Lección 3. – PyCaret y AutoML .....</b>	<b>2</b>
<b>Presentación y objetivos.....</b>	<b>2</b>
<b>1. Machine Learning visto hasta ahora.....</b>	<b>3</b>
<b>2. PyCaret y el AutoML .....</b>	<b>4</b>
<b>3. Instalación de Pycaret.....</b>	<b>6</b>
<b>4. PyCaret en Clasificación Binaria .....</b>	<b>7</b>
<b>5. Predicción en Kaggle.....</b>	<b>22</b>
<b>6. Puntos clave.....</b>	<b>23</b>

## Lección 3. – PyCaret y AutoML

### PRESENTACIÓN Y OBJETIVOS

- | En la presente lección abordaremos el aprendizaje del AutoML de la mano de PyCaret que se está convirtiendo en una de las mejores herramientas de Machine Learning a fecha Mayo-Julio 2021.
- | La misma será una introducción, y servirá como punto de partida para asignaturas centradas en Machine Learning.
- | En nuestro caso, lo explicaremos porque nos hace falta para poder resolver uno de los ejercicios prácticos en una de las lecciones.



#### **Objetivos**

- | **Conocer la tendencia en Machine Learning**
- | **Conocer PyCaret y el AutoML**
- | **Conocer Kaggle un poco mejor**

## 1. MACHINE LEARNING VISTO HASTA AHORA

Nuevamente nos vemos en la necesidad de explicar algunas cosas de Machine Learning.

Hasta ahora, hemos visto los siguientes contenidos relacionados con este tema:

- | En Creación de Aplicaciones Python explicamos algo sobre Iris Dataset para poder crear Aplicaciones con Machine Learning.
- | También, en Creación de Aplicaciones, hablamos de Series temporales para poder explicar una de las finalidades de Dash. Aunque en aquel entonces no hicimos predicciones, aunque sí que mencionamos algunas de las herramientas posibles para ello.
- | En Fundamentos de Big Data explicamos el Titanic Dataset para poder explicar conceptos básicos de Data Mining, y un montón de Gráficos.
  - Ya viendo venir que podríamos encontrarnos con la necesidad de explicar PyCaret en la presente Asignatura, se trató de condensar las lecciones 4 y 5 en el tema 4, y añadimos una lección 5 en la cual hicimos predicción en Kaggle.

En esta ocasión trataremos de hacer una breve introducción al AutoML, que es una tendencia en Inteligencia Artificial. La automatización es una realidad a día de hoy.

Como programadores, en el corto plazo, tiene sus ventajas, dado que seremos capaces de hacer en menos tiempo proyectos mucho mejor.

En el largo plazo, la automatización podría disminuir la carga de trabajo, y, por ello, de empleabilidad, aunque, en los próximos años probablemente no haga más que aumentar, porque salen proyectos nuevos cada día.

## 2. PYCARET Y EL AUTOML

Tal vez te estés preguntando qué es AutoML, y qué es PyCaret.

Para ello, te comentaremos lo siguiente..

| ¿Quiés el creador de PyCaret?

- PyCaret es creada por una comunidad de desarrolladores, liderada por Moez Ali cuyo perfil de LinkedIn es el siguiente:

[linkedin.com/in/profile-moez](https://www.linkedin.com/in/profile-moez)

| Si PyCaret, como veremos, me resuelve los proyectos en mucho menos tiempo, y con mejor grado de detalle..

*Entonces, ¿Por qué no explicarlo directamente?*

- Pudimos haber explicado directamente una herramienta de AutoML como en este caso PyCaret, pero quizá no se habría entendido que hace internamente la herramienta. Y el objetivo no es solo predecir, sino entender qué estamos haciendo.

En este caso, explicaremos pocas cosas, pero sí las necesarias para poder comprender alguno de los ejercicios prácticos que haremos.

En Machine Learning tendrá el/la alumno/a la oportunidad en un futuro de profundizar mucho más.

Puede ser por su propia cuenta, de forma autodidacta, o incluso en las asignaturas de Machine Learning.

Lo ideal es que nos acostumbremos a ver herramientas nuevas cada poco tiempo, y, si podemos, practicar con ellas.

Pudiendo hacer proyectos personales, o de empresa.

| ¿Por qué PyCaret? ¿Existen más herramientas?

- Sí, existen más herramientas de este tipo. En el caso del docente que creó este contenido, la misma fue una apuesta personal hacia Julio 2020 aproximadamente cuando éste la conoció. En aquel momento, muchísimos programadores no hablaban bien de la misma, opinando que no tenía sentido usarla. Yo no lo veía así, en absoluto.

Desde primeros de 2021..la cosa no ha hecho más que ganar popularidad y aceptación..

Por citar algunos ejemplos, podemos decir que:

- ✓ La propia Microsoft la menciona

<https://docs.microsoft.com/en-us/samples/azure/azureml-examples/automl-with-pycaret/>

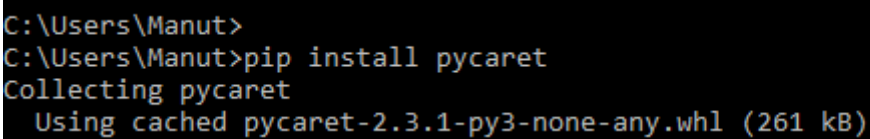
- ✓ PyCaret ofrece la posibilidad de hacer diferentes tipos de proyectos.

<https://pycaret.readthedocs.io/en/latest/tutorials.html>

- ✓ PyCaret ofrece muchas ventajas a la hora de hacer mismamente una Clasificación Binaria, pero lo ideal es que veamos todo lo que conseguimos en pocas líneas con un ejemplo práctico.

### 3. INSTALACIÓN DE PYCARET

Lo primero es instalar PyCaret, y para ello, lo ideal es hacerlo antes de arrancar nuestro Entorno de Desarrollo Jupyter.



```
C:\Users\Manut>  
C:\Users\Manut>pip install pycaret  
Collecting pycaret  
  Using cached pycaret-2.3.1-py3-none-any.whl (261 kB)
```

*Figura 3.1: Instalación de PyCaret en la "cmd"*

La instalación es muy larga, por ello, no hemos mostrado mas que las primeras líneas.

A continuación abríamos nuestro Entorno de Desarrollo, que será Jupyter, por ejemplo y trabajaremos con el Titanic Dataset.

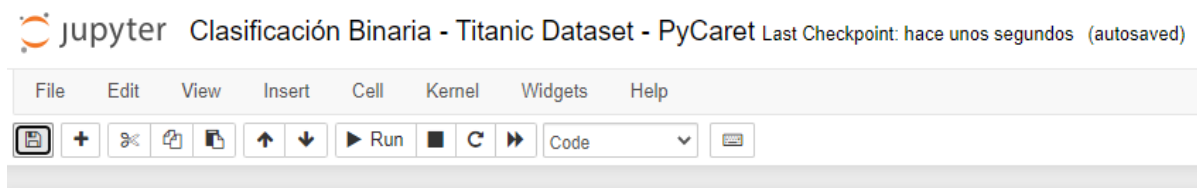
Si recuerdas, para resolver este Dataset era necesario seguir bastantes pasos, y, el código, aunque no era muy muy extenso, tampoco era muy breve.

Ahora resolveremos el mismo ejercicio usando PyCaret, y veremos cómo en pocas líneas hacemos no solo lo mismo, sino también mucho mejor.

## 4. PYCARET EN CLASIFICACIÓN BINARIA

A continuación explicamos paso a paso los pasos a seguir para resolver el Titanic Dataset con PyCaret.

La misma es una introducción, podría, y debería de hecho, optimizarse para mejorar la predicción, no obstante, para esta Asignatura es más que suficiente. Pronto tendrás oportunidad de profundizar sobre este tema.



### step 0 - Instalación de PyCaret

```
In [1]: # !pip install pycaret
# Instalarlo ANTES de abrir Jupyter, directamente en el "cmd"
# cmd: pip install pycaret
# (es lo más rápido)
```

### step 1 - Importamos nuestras Dependencias

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
In [3]: # importamos todo de PyCaret classification
from pycaret.classification import *
```

Figura 4.1: PyCaret en clasificación binaria (parte 1)



## step 2 - Importamos el Titanic Dataset

```
In [4]: # Titanic Dataset

# C:\Users\Manut\Desktop\apuntes_big_data_2\TEMA_3\datasets
# train.csv
# test.csv
# gender_submission
```

```
In [5]: df = pd.read_csv("C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_3/datasets/train.csv")
df.head()
```

```
Out[5]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figura 4.2: PyCaret en clasificación binaria (parte 2)

```
clf = setup(data = df,
            target = "Survived",
            ignore_features = ["Name", "Ticket", "PassengerId"])
```

Processing:

---

Initiated ..... 11:36:37

Status ..... Preprocessing Data

---

Following data types have been inferred automatically, if they are correct press enter to continue or type 'quit' otherwise.

---

	Data Type
Pclass	Categorical
Sex	Categorical
Age	Numeric
SibSp	Categorical
Parch	Categorical
Fare	Numeric
Cabin	Categorical
Embarked	Categorical
Survived	Label

Figura 4.3: PyCaret en clasificación binaria (parte 3)

Pulsamos Enter..para poder continuar, si estamos de acuerdo..

### step 3 - setup( )

```
In [6]: # En este caso concreto, no haríamos el drop()
# las columnas que no queremos las ignoramos con ignore_features
# la data es el propio dataframe obtenido con train.csv
# la columna (y) es "Survived"
# Si estamos de acuerdo con los tipos de datos, click en Enter.
# (Cuando ejecutamos la celda, debemos esperar y después hacer click en Enter..)
# Pclass podría ser "ordinal", pero lo podemos dejar como Categorical
# "SibSp" y "Parch" son categoricas con un limitado número de posibles valores
# pudimos considerarlas (quizá) como numéricas, no obstante

clf = setup(data = df,
            target = "Survived",
            ignore_features = ["Name", "Ticket", "PassengerId"])
```

	Description	Value
0	session_id	6052
1	Target	Survived
2	Target Type	Binary
3	Label Encoded	0: 0, 1: 1
4	Original Data	(891, 12)
5	Missing Values	True
6	Numeric Features	2
7	Categorical Features	6
8	Ordinal Features	False

Figura 4.4: PyCaret en clasificación binaria (parte 4)

La ejecución continúa..

## step 4 - compare\_models( )

```
In [7]: # k-Fold cross validation
# este concepto mejor esperar a Machine Learning por la explicación técnica
compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
<b>gbc</b>	Gradient Boosting Classifier	0.8412	0.8831	0.6966	0.8460	0.7601	0.6436	0.6534	0.2680
<b>lightgbm</b>	Light Gradient Boosting Machine	0.8332	0.8810	0.7190	0.8140	0.7560	0.6310	0.6396	0.3110
<b>rf</b>	Random Forest Classifier	0.8315	0.8830	0.7666	0.7804	0.7674	0.6358	0.6416	0.4280
<b>lr</b>	Logistic Regression	0.8250	0.8723	0.7271	0.7876	0.7510	0.6169	0.6225	1.8440
<b>et</b>	Extra Trees Classifier	0.8186	0.8622	0.7318	0.7769	0.7480	0.6070	0.6132	0.3830
<b>ridge</b>	Ridge Classifier	0.8106	0.0000	0.7134	0.7622	0.7330	0.5867	0.5910	0.0230
<b>lda</b>	Linear Discriminant Analysis	0.8074	0.8598	0.6962	0.7686	0.7249	0.5776	0.5843	0.0580
<b>ada</b>	Ada Boost Classifier	0.8042	0.8524	0.7186	0.7467	0.7263	0.5746	0.5802	0.2020
<b>dt</b>	Decision Tree Classifier	0.7946	0.7744	0.7184	0.7191	0.7155	0.5553	0.5580	0.0310
<b>knn</b>	K Neighbors Classifier	0.7047	0.7318	0.5059	0.6075	0.5488	0.3339	0.3381	0.0750
<b>svm</b>	SVM - Linear Kernel	0.6875	0.0000	0.7180	0.6163	0.6238	0.3768	0.4152	0.0310
<b>nb</b>	Naive Bayes	0.6629	0.7913	0.1152	0.7583	0.1977	0.1128	0.1940	0.0270
<b>qda</b>	Quadratic Discriminant Analysis	0.6339	0.5963	0.4561	0.5398	0.4522	0.1996	0.2010	0.0470

```
Out[7]: GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
learning_rate=0.1, loss='deviance', max_depth=3,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
```

Figura 4.5: PyCaret en clasificación binaria (parte 5)

A priori, el mejor ha sido Gradient Boosting Classifier, para utilizar el mismo en esta librería usaremos “gbc”.

Por el momento no tiene mucha importancia entender todos esos parámetros.

## step 5 - create\_model( )

(para los 2 o 3 mejores)

```
In [ ]: # Mejor en mi caso = en mi caso GBC

# Los parámetros que muestra serán explicados en Machine Learning
# de momento nos fijaremos en "Accuracy" (ya visto) y en AUC
```

Figura 4.6: PyCaret en clasificación binaria (parte 6)

```
In [8]: gbc = create_model('gbc')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8095	0.8750	0.6522	0.7895	0.7143	0.5734	0.5793
1	0.8095	0.8255	0.5652	0.8667	0.6842	0.5563	0.5824
2	0.8413	0.8918	0.6087	0.9333	0.7368	0.6303	0.6598
3	0.8548	0.8841	0.8182	0.7826	0.8000	0.6862	0.6866
4	0.8387	0.8727	0.6364	0.8750	0.7368	0.6247	0.6412
5	0.9032	0.9517	0.7727	0.9444	0.8500	0.7796	0.7882
6	0.8548	0.9420	0.7391	0.8500	0.7907	0.6804	0.6843
7	0.8548	0.8724	0.7391	0.8500	0.7907	0.6804	0.6843
8	0.7903	0.8473	0.6522	0.7500	0.6977	0.5384	0.5415
9	0.8548	0.8685	0.7826	0.8182	0.8000	0.6862	0.6866
Mean	0.8412	0.8831	0.6966	0.8460	0.7601	0.6436	0.6534
SD	0.0303	0.0365	0.0801	0.0599	0.0510	0.0700	0.0675

Figura 4.7: PyCaret en clasificación binaria (parte 7)

Creamos el modelo con Gradient Boosting Classifier (gbc), y el promedio que nos sale de Accuracy para esos intentos es 0.8412, tal y como me indicaba la tabla anterior.

Son resultados mejorables, pero para un ejercicio introducción es suficiente.

También crearemos el modelo para las otras 2 opciones de los mejores resultados.

```
In [9]: lightgbm= create_model('lightgbm')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8095	0.8630	0.6957	0.7619	0.7273	0.5814	0.5828
1	0.7778	0.8234	0.4783	0.8462	0.6111	0.4719	0.5095
2	0.8413	0.8815	0.6087	0.9333	0.7368	0.6303	0.6598
3	0.8871	0.9006	0.8182	0.8571	0.8372	0.7509	0.7513
4	0.8871	0.8824	0.7273	0.9412	0.8205	0.7401	0.7532
5	0.8387	0.9455	0.8182	0.7500	0.7826	0.6548	0.6564
6	0.8065	0.8997	0.7826	0.7200	0.7500	0.5926	0.5939
7	0.8226	0.8551	0.7391	0.7727	0.7556	0.6164	0.6168
8	0.8065	0.8462	0.7391	0.7391	0.7391	0.5853	0.5853
9	0.8548	0.9130	0.7826	0.8182	0.8000	0.6862	0.6866
Mean	0.8332	0.8810	0.7190	0.8140	0.7560	0.6310	0.6396
SD	0.0340	0.0338	0.0997	0.0750	0.0599	0.0784	0.0735

Figura 4.8: PyCaret en clasificación binaria (parte 8)

```
In [10]: rf = create_model('rf')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7619	0.8652	0.6957	0.6667	0.6809	0.4911	0.4914
1	0.8413	0.8717	0.6522	0.8824	0.7500	0.6375	0.6531
2	0.8413	0.8783	0.7391	0.8095	0.7727	0.6512	0.6528
3	0.8387	0.8534	0.8636	0.7308	0.7917	0.6616	0.6677
4	0.8548	0.9216	0.6364	0.9333	0.7568	0.6585	0.6830
5	0.8387	0.9301	0.8182	0.7500	0.7826	0.6548	0.6564
6	0.8387	0.9136	0.8261	0.7600	0.7917	0.6605	0.6620
7	0.8065	0.8211	0.7391	0.7391	0.7391	0.5853	0.5853
8	0.8548	0.8857	0.8261	0.7917	0.8085	0.6917	0.6921
9	0.8387	0.8896	0.8696	0.7407	0.8000	0.6663	0.6723
Mean	0.8315	0.8830	0.7666	0.7804	0.7674	0.6358	0.6416
SD	0.0264	0.0314	0.0815	0.0740	0.0359	0.0547	0.0570

Figura 4.9: PyCaret en clasificación binaria (parte 9)

De igual manera que en el caso anterior, hacemos el `tune_model()` para los 3 mejores, y le podemos decir que sea optimizado por AUC o por Accuracy.

Ambas son formas de optimizar el modelo, aunque existen más.

De momento no explicaremos que es AUC para que no se acumulen los conceptos, probablemente sea explicado en asignaturas de Machine Learning.

## step 6 - `tune_model()` para los 2-3 mejores

```
In [11]: # gbc optimizado con AUC
tune_gbc_auc = tune_model(gbc, optimize="AUC")
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.6349	0.8636	0.0000	0.0000	0.0000	0.0000	0.0000
1	0.6349	0.8522	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.6349	0.8946	0.0000	0.0000	0.0000	0.0000	0.0000
3	0.6452	0.8642	0.0000	0.0000	0.0000	0.0000	0.0000
4	0.6452	0.8892	0.0000	0.0000	0.0000	0.0000	0.0000
5	0.6452	0.9369	0.0000	0.0000	0.0000	0.0000	0.0000
6	0.6290	0.9041	0.0000	0.0000	0.0000	0.0000	0.0000
7	0.6290	0.8774	0.0000	0.0000	0.0000	0.0000	0.0000
8	0.6290	0.8389	0.0000	0.0000	0.0000	0.0000	0.0000
9	0.6290	0.8986	0.0000	0.0000	0.0000	0.0000	0.0000
Mean	0.6356	0.8820	0.0000	0.0000	0.0000	0.0000	0.0000
SD	0.0067	0.0272	0.0000	0.0000	0.0000	0.0000	0.0000

Figura 4.10: PyCaret en clasificación binaria (parte 10)

Por algún motivo me imprime muchas columnas de 0 que no deberían estar ahí, pero no le daremos importancia, a veces es algo que ocurre.

Pudiéramos igualmente, ejecutar la celda nuevamente.

```
In [12]: # gbc optimizado con Accuracy
tune_gbc_acc = tune_model(gbc, optimize="Accuracy")
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7937	0.8701	0.6957	0.7273	0.7111	0.5507	0.5511
1	0.7937	0.8473	0.5217	0.8571	0.6486	0.5145	0.5463
2	0.8571	0.8799	0.6522	0.9375	0.7692	0.6705	0.6937
3	0.8548	0.9006	0.8636	0.7600	0.8085	0.6924	0.6961
4	0.8065	0.8585	0.5455	0.8571	0.6667	0.5396	0.5670
5	0.8548	0.9426	0.7727	0.8095	0.7907	0.6797	0.6801
6	0.8548	0.9197	0.7826	0.8182	0.8000	0.6862	0.6866
7	0.8065	0.8161	0.7391	0.7391	0.7391	0.5853	0.5853
8	0.7903	0.8200	0.7391	0.7083	0.7234	0.5547	0.5550
9	0.8710	0.9080	0.8261	0.8261	0.8261	0.7235	0.7235
Mean	0.8283	0.8763	0.7138	0.8040	0.7483	0.6197	0.6285
SD	0.0309	0.0399	0.1067	0.0674	0.0577	0.0737	0.0691

Figura 4.11: PyCaret en clasificación binaria (parte 11)

```
In [13]: # lightgbm optimizado con AUC
tune_lightgbm_auc = tune_model(lightgbm, optimize="AUC")
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8254	0.8793	0.6957	0.8000	0.7442	0.6126	0.6161
1	0.7778	0.8310	0.4783	0.8462	0.6111	0.4719	0.5095
2	0.8413	0.8696	0.6087	0.9333	0.7368	0.6303	0.6598
3	0.8710	0.8795	0.8636	0.7917	0.8261	0.7238	0.7256
4	0.8548	0.9142	0.6364	0.9333	0.7568	0.6585	0.6830
5	0.8871	0.9580	0.8182	0.8571	0.8372	0.7509	0.7513
6	0.8226	0.9231	0.7391	0.7727	0.7556	0.6164	0.6168
7	0.8387	0.8807	0.7391	0.8095	0.7727	0.6481	0.6497
8	0.7903	0.8551	0.6522	0.7500	0.6977	0.5384	0.5415
9	0.8710	0.8974	0.7391	0.8947	0.8095	0.7133	0.7207
Mean	0.8380	0.8888	0.6970	0.8389	0.7548	0.6364	0.6474
SD	0.0334	0.0343	0.1045	0.0618	0.0629	0.0806	0.0748

Figura 4.12: PyCaret en clasificación binaria (parte 12)

```
In [14]: # lightgbm optimizado con Accuracy
tune_lightgbm_acc = tune_model(lightgbm, optimize="Accuracy")
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7778	0.8587	0.6957	0.6957	0.6957	0.5207	0.5207
1	0.8413	0.8380	0.6087	0.9333	0.7368	0.6303	0.6598
2	0.8571	0.8913	0.6522	0.9375	0.7692	0.6705	0.6937
3	0.8710	0.8875	0.8636	0.7917	0.8261	0.7238	0.7256
4	0.8871	0.8972	0.7273	0.9412	0.8205	0.7401	0.7532
5	0.8548	0.9443	0.8636	0.7600	0.8085	0.6924	0.6961
6	0.8387	0.9075	0.8261	0.7600	0.7917	0.6605	0.6620
7	0.7903	0.8573	0.6522	0.7500	0.6977	0.5384	0.5415
8	0.8065	0.8094	0.7826	0.7200	0.7500	0.5926	0.5939
9	0.9032	0.9186	0.9130	0.8400	0.8750	0.7963	0.7981
Mean	0.8428	0.8810	0.7585	0.8129	0.7771	0.6565	0.6644
SD	0.0388	0.0382	0.1006	0.0892	0.0552	0.0835	0.0849

Figura 4.13: PyCaret en clasificación binaria (parte 13)

```
In [15]: # lightgbm optimizado con Accuracy
tune_rf_auc = tune_model(rf, optimize="AUC")
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7937	0.8663	0.7826	0.6923	0.7347	0.5669	0.5697
1	0.8571	0.8560	0.6522	0.9375	0.7692	0.6705	0.6937
2	0.8571	0.8826	0.6957	0.8889	0.7805	0.6769	0.6881
3	0.8226	0.8506	0.8182	0.7200	0.7660	0.6240	0.6273
4	0.8710	0.8881	0.7727	0.8500	0.8095	0.7123	0.7141
5	0.8548	0.9415	0.8636	0.7600	0.8085	0.6924	0.6961
6	0.8387	0.9064	0.8261	0.7600	0.7917	0.6605	0.6620
7	0.8387	0.8629	0.7826	0.7826	0.7826	0.6544	0.6544
8	0.7742	0.8462	0.7826	0.6667	0.7200	0.5328	0.5376
9	0.8710	0.9041	0.8261	0.8261	0.8261	0.7235	0.7235
Mean	0.8379	0.8805	0.7802	0.7884	0.7789	0.6514	0.6567
SD	0.0308	0.0286	0.0602	0.0824	0.0315	0.0580	0.0586

Figura 4.14: PyCaret en clasificación binaria (parte 14)



```
In [16]: # lightgbm optimizado con Accuracy
tune_rf_acc = tune_model(rf, optimize="Accuracy")
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7937	0.8663	0.7826	0.6923	0.7347	0.5669	0.5697
1	0.8571	0.8560	0.6522	0.9375	0.7692	0.6705	0.6937
2	0.8571	0.8826	0.6957	0.8889	0.7805	0.6769	0.6881
3	0.8226	0.8506	0.8182	0.7200	0.7660	0.6240	0.6273
4	0.8710	0.8881	0.7727	0.8500	0.8095	0.7123	0.7141
5	0.8548	0.9415	0.8636	0.7600	0.8085	0.6924	0.6961
6	0.8387	0.9064	0.8261	0.7600	0.7917	0.6605	0.6620
7	0.8387	0.8629	0.7826	0.7826	0.7826	0.6544	0.6544
8	0.7742	0.8462	0.7826	0.6667	0.7200	0.5328	0.5376
9	0.8710	0.9041	0.8261	0.8261	0.8261	0.7235	0.7235
Mean	0.8379	0.8805	0.7802	0.7884	0.7789	0.6514	0.6567
SD	0.0308	0.0286	0.0602	0.0824	0.0315	0.0580	0.0586

Figura 4.15: PyCaret en clasificación binaria (parte 15)

Ahora haremos varias gráficas..

## step 7 - plot\_model( ) para el mejor

```
In [17]: # tune_lightgbm_acc consigue
# 0.84 (Accuracy) - 0.88 (AUC)
plot_model(tune_lightgbm_acc, plot = 'error')
```

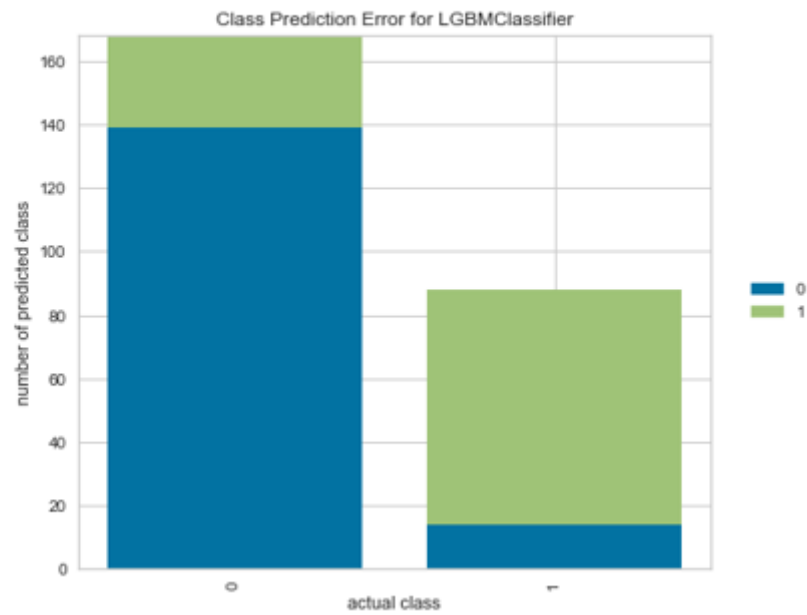


Figura 4.16: PyCaret en clasificación binaria (parte 16)

```
In [18]: plot_model(tune_lightgbm_acc, plot = 'pr')
```

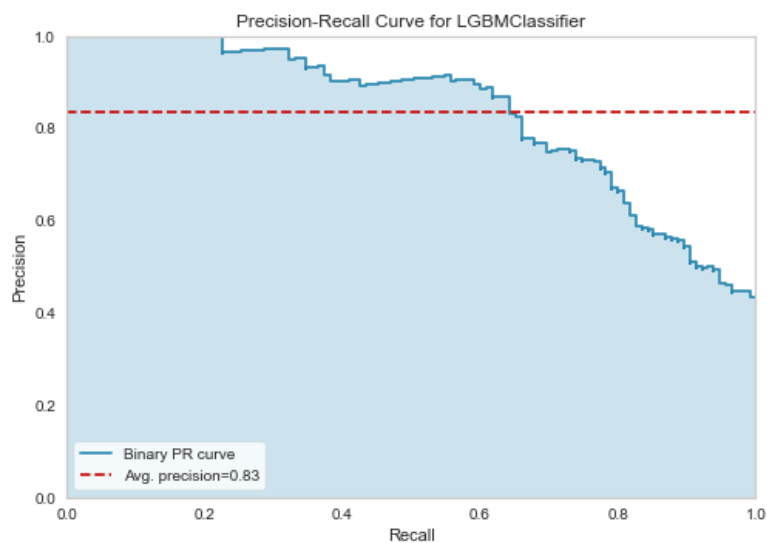


Figura 4.17: PyCaret en clasificación binaria (parte 17)

```
In [19]: plot_model(tune_lightgbm_acc, plot='feature')
```

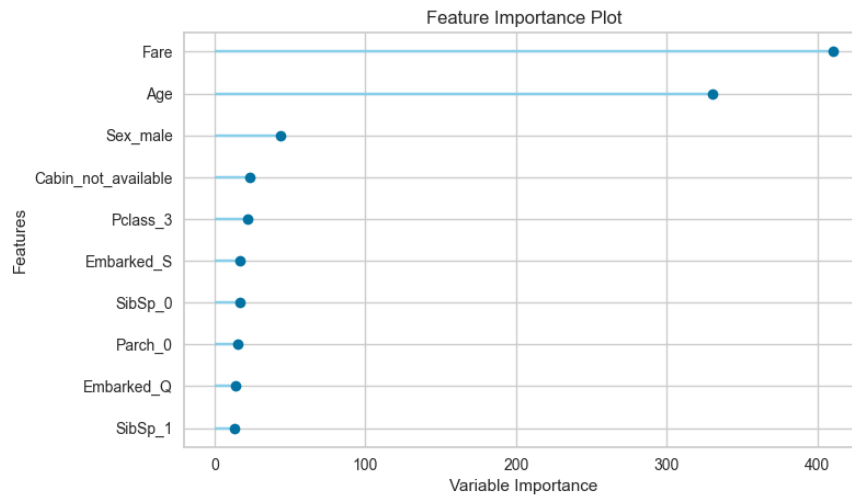


Figura 4.18: PyCaret en clasificación binaria (parte 18)

```
In [20]: plot_model(tune_lightgbm_acc, plot = 'confusion_matrix')
```

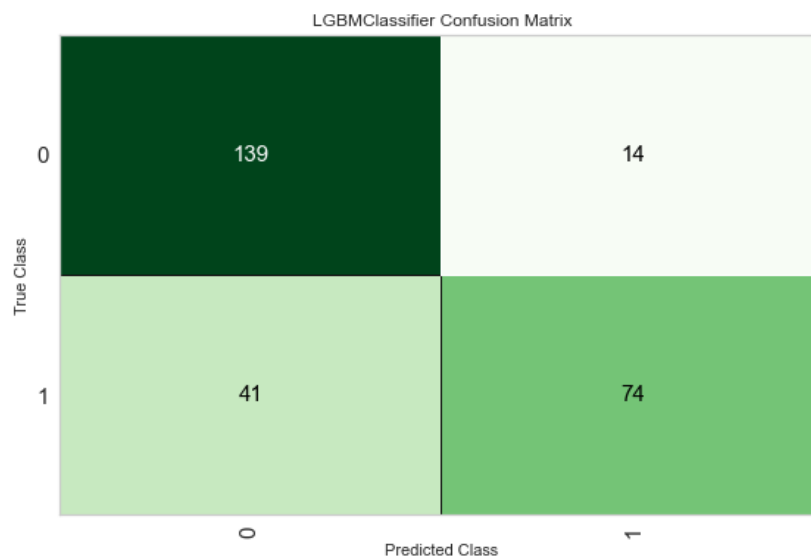


Figura 4.19: PyCaret en clasificación binaria (parte 19)

En este caso tenemos muchísimas gráficas solo con una instrucción podemos ver todas una a una

## step 8 - evaluate\_model( ) para el mejor

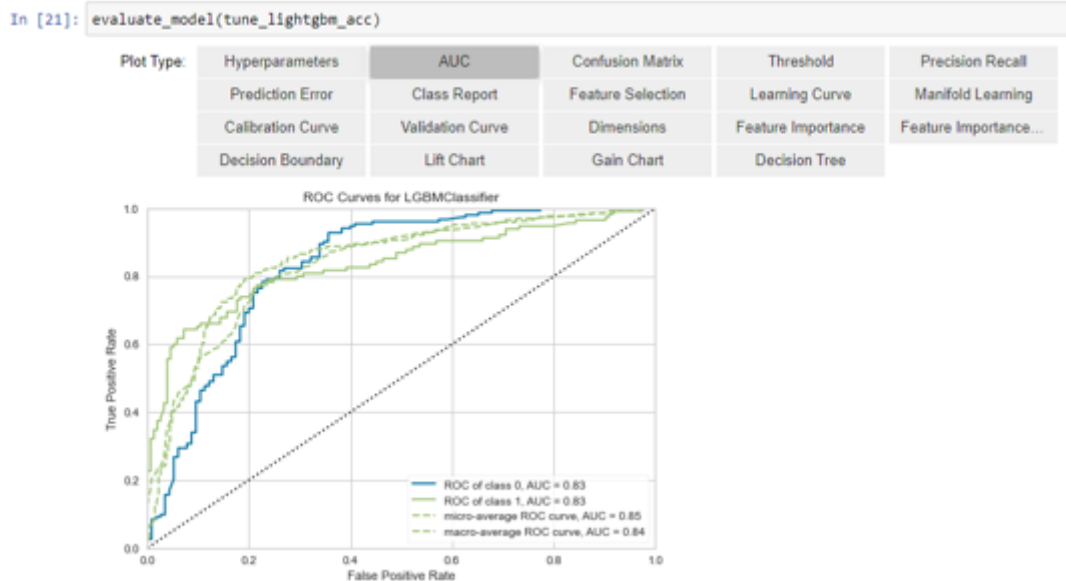


Figura 4.20: PyCaret en clasificación binaria (parte 20)

## step 9 - nos vamos a test.csv



Figura 4.21: PyCaret en clasificación binaria (parte 21)

## step 10 - hacemos predicciones con el mejor

```
In [23]: predicciones = predict_model(tune_lightgbm_acc, data = test)
predicciones
```

```
Out[23]:
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Label	Score
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	0	0.8847
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	0	0.9572
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	0	0.9876
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	0	0.7423
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	0	0.7401
...	...	...	...	...	...	...	...	...	...	...	...	...	...
413	1305	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	NaN	S	0	0.9786
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	106.9000	C105	C	1	0.9916
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S	0	0.9855
416	1308	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN	S	0	0.9786
417	1309	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	NaN	C	0	0.6196

418 rows x 13 columns

Figura 4.22: PyCaret en clasificación binaria (parte 22)

```
In [24]: predicciones_submission = predicciones[["PassengerId", "Label"]]
predicciones_submission.head()
```

```
Out[24]:
```

	PassengerId	Label
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0

Figura 4.23: PyCaret en clasificación binaria (parte 23)

## step 11 - submissions

```
In [25]: # compruebo el formato que me piden
submission = pd.read_csv("C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_3/datasets/gender_submission.csv")
submission.head()
# por lo cual vemos que cumple con el formato
```

```
Out[25]:
```

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

Figura 4.24: PyCaret en clasificación binaria (parte 24)

```
In [26]: submission["PassengerId"] = predicciones_submission["PassengerId"]
submission["Survived"] = predicciones_submission["Label"]
submission.head()
```

```
Out[26]:
```

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0

Figura 4.25: PyCaret en clasificación binaria (parte 25)

## step 12 - creo el archivo csv para Kaggle

```
In [27]: # guardo el archivo en una ruta donde lo pueda tener localizado
# C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_3/predicciones
submission.to_csv('C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_3/predicciones/titanic_1.csv', index=False)
```

```
In [28]: # ...
# Ahora deberíamos ir a Kaggle para enviar nuestros resultados..
# Si el resultado que te sale en Kaggle es inferior al obtenido es normal.
# Toma la Asignatura de Machine Learning y practica "un poco" más para mejorarlo..
# ...
```

Figura 4.26: PyCaret en clasificación binaria (parte 26)

## 5. PREDICCIÓN EN KAGGLE

Tomo el `titanic_1.csv` y lo llevo a Kaggle, tal y como explicamos en “Fundamentos de Big Data”.

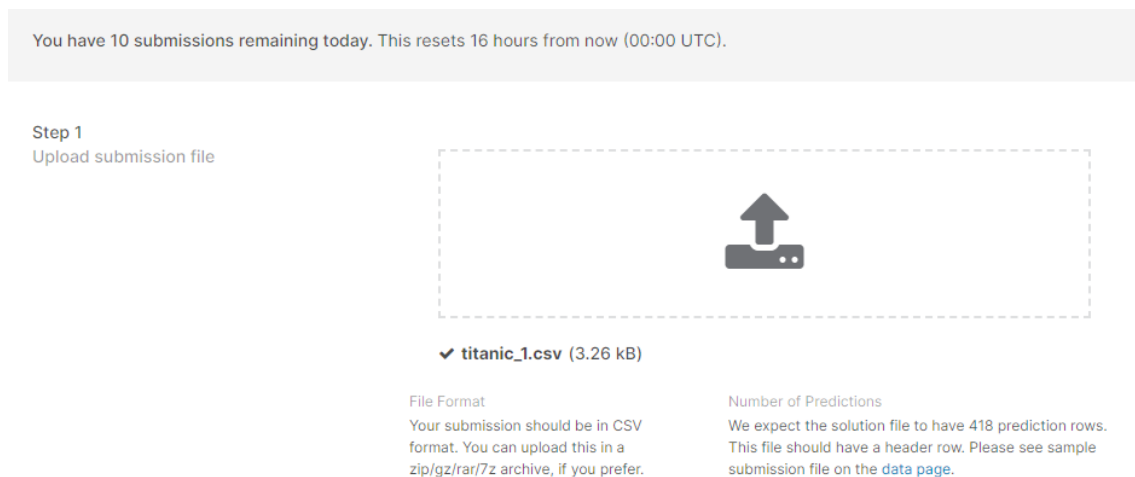


Figura 5.1: Predicción en Kaggle

- Una vez Kaggle nos devuelva el resultado, tal y como vimos en la Asignatura de Fundamentos de Big Data, puede ocurrir que el resultado sea peor que la anterior vez, y así nos lo comunicará, puede ser mejor que el anterior, y mejoraremos, quizá, nuestra posición. A partir de ahora ya es cuestión de ir mejorando poco a poco.
- Se aprende a base de practicar, y de estudiar cosas relacionadas con Matemáticas, Procesamiento del Lenguaje Natural (NLP), y Algoritmos, entre otras cosas.
- Esperamos que la presente lección te parezca interesante, y, en un futuro, puedas practicar y aprender mucho más.
- Tal vez, tras terminar las materias relacionadas con Machine Learning te sientas más seguro.

## 6. PUNTOS CLAVE

- | AutoML es una tendencia en Inteligencia Artificial
- | PyCaret se presenta como una de las mejores alternativas para el uso en Machine Learning
- | Kaggle es una herramienta IMPRESCINDIBLE para la formación en Data Science, con retos diversos, y que nos permitirá aprender muchas cosas



